Project

Detection and calculation of distance to road signs

Objective

The aim of the project is to create an application that allows detecting road signs from images captured by the camera and calculating in real time the distance from the camera to the signs.

Destiny

The application is designed for the *UP Squared AI Edge X minicomputer* . A neural network based on the *YOLO architecture is responsible for character detection* . The application distinguishes several defined types of the most common characters. The application will be created using the *Python 3 language* .

1. Preparing the project

   The preparation of the project began with the creation of a set of training data. At the beginning, a ready set was searched on the Internet , but no suitable set with Polish characters was found. In that case, the dataset was prepared independently. For this purpose, five approximately 30-minute videos recorded with a dash cam were downloaded from YouTube, and they also recorded their own using the camera included in the UP Squared AI Vision X Developer Kit. Then, with the help of the VLC program, about a thousand frames with visible road signs were extracted from the films.

   Input data description

   The next stage was the annotation of video frames, which consisted in marking the location of signs of a given type on them. Many free annotation programs were tested, finally the web application " roboflow " was chosen because of its accessible and simple user interface. All characters that are visible to a human were marked, which resulted in many relatively small characters in the data set. On average, three road signs were marked on one image frame. Data augmentation capabilities available in " roboflow " were also used to triple the size of the training set.

   Network training

After the dataset was created, the neural network was trained. The YOLOv5 network in the nano version was selected for the task due to its speed and relatively good precision. A script attached to the network repository was used to train the network. The learning itself was carried out on a GPU provided by Kaggle and lasted about 10 hours. YOLOv5 networks in non- nano versions were also trained to later compare their performance and accuracy.

Network conversion

Next, the trained network had to be prepared in such a way that it could be used with the Intel Movidius accelerator Myriad X. Initially, the neural network with trained weights was written in the format used in the PyTorch library , but this library was unable to use the accelerator. The only tool that can handle the Intel Movidius accelerator Myriad X is OpenVINO . It was necessary to convert the network to a format that is supported by this tool. The development version of the OpenVINO tool provides a script that allows you to convert a network saved in the ONNX format to a format supported by OpenVINO .
The conversion was in two stages: from PyTorch format to ONNX format using a script located in the YOLOv5 repository, then from ONNX to a format supported by OpenVINO .

Problems with OpenVINO

After converting the neural network with weights to the appropriate version, a problem with accelerator support was encountered. It turned out that the development version of OpenVINO , which can only be installed via the PyPI tool , does not support Intel Movidius accelerators Myriad X. There was no mention of this in the documentation. One way to use OpenVINO developer tools and Intel Movidius accelerators at the same time is to install two versions of OpenVINO .
The first version with only the OpenVINO engine , without development tools, must be installed using the offline installer, because only it installs the appropriate drivers and libraries to support Intel Movidius accelerators Myriad . Then, in order to use the development tools, install a second copy of OpenVINO , this time using the PyPI tool . It's best to do this in a virtual environment so there are no issues with environment variables. When we already have two installations of the OpenVINO tool (one supports accelerators, but does not have development tools, the other has a development tool, but does not support accelerators) we can choose which instance is to be currently used. This can be done by modifying environment variables.

Interface execution

The next step in the project is to create an interface that will allow the use of a neural network in the traffic sign detection algorithm. Before submitting data to the network, the image frame must be subjected to appropriate processing. From the resulting vector, data on the position of the characters, their type and certainty of detection should be read. The Non-maximum Suppression technique was used . There are many examples of such interfaces on the Internet for the YOLOv5 network, but no example that could use Intel

Movidius accelerators was found Myriad . Therefore, an appropriate interface was designed based on the one used in the detection script located in the YOLOv5 repository. After implementing the interface, a neural network was launched using the Intel Movidius accelerator Myriad X and tests were carried out to recognize and locate road signs on a single image frame.

2. Character detection algorithm

The traffic sign detection algorithm used in the work uses the YOLOv5 neural network. This network allows the detection of objects in the image in real time. It was learned on a set of 700 frames, cut from recordings with car cameras, which show road signs along with annotations about their location and category. The algorithm divides the detected characters into the following categories:

- Informative
- Informative additional
- Informative large
- Warrant
- Priority
- Prohibition
- Big ban
- Stop
- Warning
- Additional warnings
- Give way

Distances are not calculated for additional information signs, large information signs, large prohibition signs and additional warning signs, which is caused by their irregular shapes.

3. Character detection accuracy and performance

The algorithm uses YOLOv5 nano version , it is the smallest and fastest available YOLOv5 networks. Its accuracy when taught is 81% with 50% confidence and 60% with 95% confidence. Due to the fact that the characters on the frames may be small, the network was trained on frames with a relatively large size of 1280 px , thanks to which its precision is much higher than if it was trained on frames scaled to only 640 px (at 640 px precision for 50% confidence is only 63%).
Processing time per 1280 px frame on the UP Squared AI Edge X mini PC using the built-in Intel Movidius accelerator Myriad X is around 0.25 seconds. When using the CPU, the processing time increases to about 0.45 seconds. Scaling the frames to the size of 640 px shortens the frame processing time four times, but the precision is definitely lower.

4. Calculation of the distance from the mark

Each frame of the film is fed to the input of the neural network. The network returns the upper left and lower right points that are part of the rectangle inside which the character is

located. It also returns the number of the class the character belongs to. Then the distance to a given mark is calculated from the following formula:

$$d = \frac{f * s_r * R}{s_i * P}$$

Where:

d – distance from the camera to the given mark (mm)
f - camera focal length (mm)
sr - actual character size (mm)
R - image resolution
si - size of the character in the photo (px)
P – sensor size (mm)

The actual size of a character is always the shorter side of the character. The resolution and size of the sensor are adjusted to the current shorter side, i.e. if the shorter side is in the x-axis, the width of the image is taken as the resolution and width of the sensor.
After determining the distance to the mark, it is converted to meters and marked on the frame together with the rectangles inside which the mark is located and information about the type of the detected mark. An example frame is shown in Figure 1.

Figure 1

5. Installation and use of the application on the UP Squared AI Edge X

Step 1 - Install *Ubuntu 20.04* on the *UP Squared AI Edge X mini PC*

1. Download *Ubuntu 20.04* - https://releases.ubuntu.com/20.04/
2. Install *Ubuntu 20.04* on a mini PC by following the guide - https://ubuntu.com/tutorials/install-ubuntu-desktop , in the option to choose the type of installation, choose minimal installation

Step 2 - Installing the traffic sign detection app

1. After logging in, open the console (use the keyboard shortcut Ctrl + Alt + t)
2. Invoke (type the command, click Enter and wait for it to execute) the following commands in turn:

a. `sudo apt update -y`
b. `sudo apt upgrade -y`

3. Restart the mini PC and after logging in, open the console

4. Run the following commands one by one:

```
a. sudo apt install git pip wget -y
b. python3 -m pip install --upgrade pip
c. cd ~/Downloads/
d. wget
https://registrationcenterdownload.intel.com/akdlm/irc_nas/18617/l_openvino
_toolkit_p_2022.1.0.643_offline.sh
e. sudo chmod +x l_openvino_toolkit_p_2022.1.0.643_offline.sh
f. sudo ./l_openvino_toolkit_p_2022.1.0.643_offline.sh
```

5. Install the *OpenVINO software* using the installation assistant, leave the default options, ignore any messages about missing packages (graphical instructions)

6. Reopen the console

7. Run the following commands one by one:

```
a. sudo -E/opt/intel/openvino_2022/install_openvino_dependencies.sh
b. echo 'source /opt/intel/openvino_2022/setupvars.sh' >>~/. bashrc
c. exec bash
d.
/opt/intel/openvino_2022/runtime/3rdparty/hddl/install_IVAD_VPU_dependencie
s.sh
```

8. Restart the mini PC and after logging in, open the console

9. Run the following commands one by one:

```
a. git clone https://github.com/Remni1/ASB-SignLocationUP2.git
b. cd ASB-SignLocation-UP2
c. pip install -r requirements.txt
```

Step 3 - Using the application

The application is launched using the *Python interpreter* . In addition, the operation of the script can be configured by users through the following options:

• -- use_cam – specifies that the image is to be captured from the camera
• -- live_display – determines whether the result of the program's operation is to be displayed live while it is running
• -- save_video – determines whether the result of the program operation is to be saved on the disk
• - frame_rate – determines how many frames the prediction is to be made
• - video_path – specifies the path to the video file to be processed
• - fps – defines the desired number of frames per second

6. Repositories

Link to the code repository:
https://github.com/Remni1/ASB-SignLocation-UP2

Dataset link:
https://drive.google.com/file/d/1fKvF2E6mOlJrhUvy8w8Vrq6AHkGXhHZR/view?usp=sharing